Deep Learning: Homework 2

Author: Rudy Martinez

Date: 9/21/2021

Google Colab Link: https://colab.research.google.com/drive/10mGfwhVDcCXcHs4jQHAFrkIM_AVgplBp?usp=sharing

Libraries

```
#Data Maneuvering
import pandas as pd
import numpy as np

#Data Visualization
import matplotlib.pyplot as plt

#Machine Learning
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
```

Part 1 - Write a Python code in Colab using NumPy, Panda, Scikit-Learn to complete the following tasks:

- **(1)** Import the `Auto MPG` dataset with pandas.read_csv() using the dataset URL, use the attribute names as explained in the dataset description as the column names (5pt), view the strings '?' as the missing value, and whitespace (i.e., '\s+') as the column delimiter. Print out the shape and first 5 rows of the obtained DataFrame. (5pt)

```
#Import Data
data = pd.read_csv("http://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/auto-mpg.data", delimiter="\s+",
                   names = ["mpg", "cylinders", "displacement", "horsepower",
                            "weight", "acceleration", "model_year", "origin",
                            "car_name"])

#Missing Values Identifier
missing_values = ['?', ' ?', '? ', ' ? ']

#Replace ? with NaN
auto_df = data.replace(missing_values, np.nan)
```

```
#Shape of the Obtained Dataframe
auto_df.shape
```

```
(398, 9)
```

```
#First Five Rows of the Obtained Dataframe
auto_df.head()
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	car_name
0	18.0	8	307.0	130.0	3504.0	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693.0	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150.0	3436.0	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150.0	3433.0	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140.0	3449.0	10.5	70	1	ford torino

- **(2)** Delete the “car_name” column using .drop() method as it is irrelevant to the prediction. Print out a concise summary of the new DataFrame using .info() and check if NULL value exists in each column (5pt)

```
#Drop `car_name` from the Dataframe
auto_df = auto_df.drop("car_name", axis = 1)
```

```
#Summary View
print(auto_df.info(), "\n")

#Reveals Sum of NaN in Dataframe
print("----------------\nNull Values\n\n", auto_df.isnull().sum())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
Data columns (total 8 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   mpg           398 non-null    float64
 1   cylinders     398 non-null    int64  
 2   displacement  398 non-null    float64
 3   horsepower    392 non-null    object 
 4   weight        398 non-null    float64
 5   acceleration  398 non-null    float64
 6   model_year    398 non-null    int64  
 7   origin        398 non-null    int64  
dtypes: float64(4), int64(3), object(1)
memory usage: 25.0+ KB
None

----------------
Null Values

 mpg             0
```

```
cylinders       0
displacement    0
horsepower      6
weight          0
acceleration    0
model_year      0
origin          0
dtype: int64
```

- **(3)** Replace the NULL value with the mean value of the column using .fillna(). Print out the concise summary of the new DataFrame and recheck if NULL value exists in each column (5pt)

```
#Convert `horsepower` Column to float64
auto_df['horsepower'] = auto_df['horsepower'].astype('float64')

#Fill NULL with Mean Value
auto_df['horsepower'].fillna(auto_df['horsepower'].mean(), inplace = True)
```

```
#Summary View
print(auto_df.info(), "\n")

#Reveals Sum of NaN in Dataframe
print("----------------\nNull Values\n\n", auto_df.isnull().sum())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
Data columns (total 8 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   mpg           398 non-null    float64
 1   cylinders     398 non-null    int64
 2   displacement  398 non-null    float64
 3   horsepower    398 non-null    float64
 4   weight        398 non-null    float64
 5   acceleration  398 non-null    float64
 6   model_year    398 non-null    int64
 7   origin        398 non-null    int64
dtypes: float64(5), int64(3)
memory usage: 25.0 KB
None

----------------
Null Values

 mpg             0
cylinders       0
displacement    0
horsepower      0
weight          0
acceleration    0
model_year      0
```

```
origin          0
dtype: int64
```

- **(4)** For the 'origin' column with categorical attribute, replace it with the columns with numerical attributes using one-hot encoding (you can use either get_dummies() in Pandas or OneHotEncoder in Scikit-Learn). Print out the first 5 rows of the newly obtained DataFrame. (10pt)

```
#Checking the # of Unique Values in the `origin` column
auto_df["origin"].unique()
```

```
array([1, 3, 2])
```

```
#Get Dummies
auto_df = pd.get_dummies(data = auto_df, columns = ["origin"])
auto_df.head()
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin_1	origin_2	origin_3
0	18.0	8	307.0	130.0	3504.0	12.0	70	1	0	0
1	15.0	8	350.0	165.0	3693.0	11.5	70	1	0	0
2	18.0	8	318.0	150.0	3436.0	11.0	70	1	0	0
3	16.0	8	304.0	150.0	3433.0	12.0	70	1	0	0
4	17.0	8	302.0	140.0	3449.0	10.5	70	1	0	0

- **(5)** Learn a linear regression model (fit_intercept=True) to predict the "mpg" column from the remaining columns in the obtained DataFrame of Step 4.
  - Separate the "mpg" column from other columns and view it as the label vector and others as the feature matrix (5pt)
  - Split the data into a training set (80%) and testing set (20%) using train_test_split and print out their shapes (5pt)
  - Train the model using the training set and print out the coefficients of the model (5 pt)
  - Use the learned model to predict on the test set and print out the mean squared error of the predictions (5pt)

```
#Separate `mpg` and Other Columns
feature_matrix = auto_df.drop("mpg", axis=1)
label_vector = auto_df['mpg']

#Split the Data Using train_test_split
X_train, X_test, y_train, y_test = train_test_split(feature_matrix, label_vector, test_size = 0.2, random_state = 0)

#Print out Shapes
print("X_train Shape: ", X_train.shape)
print("y_train Shape: ", y_train.shape)
print("X_test Shape: ", X_test.shape)
```

```
print("y_test Shape: ", y_test.shape)
```

```
X_train Shape:  (318, 9)
y_train Shape:  (318,)
X_test Shape:  (80, 9)
y_test Shape:  (80,)
```

```
#Train (Fit) the Linear Regression Model
LinReg = LinearRegression(fit_intercept=True)
LinReg.fit(X_train, y_train)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

```
#Predict on the Test Set
y_pred = LinReg.predict(X_test)
```

```
#Print Coeffecients of the Model
print('Model Coefficients: \n', LinReg.coef_)

# The coefficient of determination: 1 is perfect prediction
print('\n Coefficient of determination (r2 Score): %.2f'
      % round(r2_score(y_test, y_pred), 2))
```

```
Model Coefficients:
 [-0.40247363  0.02517027 -0.02090317 -0.00666591  0.19107804  0.76544926
 -1.92519814  0.72404785  1.20115029]

 Coefficient of determination (r2 Score): 0.83
```

```
#Print MSE of the Predictions
print("Mean Squared Error: ", round(mean_squared_error(y_test, y_pred), 2))
```

```
Mean Squared Error:  10.99
```

Part 2 - Write a Python code in Colab using NumPy, Panda, Scikit-Learn to complete the following tasks:

- **(1)** Import the red wine dataset with pandas.read_csv() using the dataset URL, use the semi-colon as the column delimiter, and print out both the first five rows and a concise summary of the obtained DataFrame. (10 pt)

```
#Read in Data
wine_df = pd.read_csv("http://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv", delimiter=";")
```

```
#First Five Rows
wine_df.head()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5

```
#Summary of Dataframe
wine_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         1599 non-null   float64
 1   volatile acidity      1599 non-null   float64
 2   citric acid           1599 non-null   float64
 3   residual sugar        1599 non-null   float64
 4   chlorides             1599 non-null   float64
 5   free sulfur dioxide   1599 non-null   float64
 6   total sulfur dioxide  1599 non-null   float64
 7   density               1599 non-null   float64
 8   pH                    1599 non-null   float64
 9   sulphates             1599 non-null   float64
 10  alcohol               1599 non-null   float64
 11  quality               1599 non-null   int64  
dtypes: float64(11), int64(1)
memory usage: 150.0 KB
```

- **(2)** Suppose we want to predict the quality of wine from other attributes. Divide the data into a label vector and a feature matrix. Then split them into a training set (80%) and testing set (20%) using train_test_split. (5pt)

```
feature_matrix = wine_df.drop("quality", axis=1)
label_vector = wine_df["quality"]

X_train, X_test, y_train, y_test = train_test_split(feature_matrix, label_vector, test_size = 0.2, random_state = 46)
```

- **(3)** Use the StandardScalar() in Scikit-Learn to preprocess the feature matrices of both training set and testing set. Note that the testing set can only be scaled by the mean and standard deviation values obtained from the training set. (5 pt)

```
#Instantiate Scaler
scaler = StandardScaler()

#Transform X_train and X_test
```

```
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

- **(4)** Use the preprocessed training dataset to learn a classification model with RandomForestClassifier (with 300 trees) in Scikit-Learn. Use 5-fold cross-validation to train and cross-validate the model. Print out the accuracies returned by the five folds as well as their average and standard deviation values. (10 pt)

```
#Build RandomForest Classifier
clf = RandomForestClassifier(n_estimators = 300, random_state=46)

accuracies = cross_val_score(estimator=clf, X = X_train, y = y_train, cv=5)

#Print out Accuracies
print("Accuracies: ", accuracies, "\n\n")

#Print out Average and Standard Deviation Values
print("Mean: ", accuracies.mean(), "\n\n")
print("Standard Deviation: ", accuracies.std())
```

```
Accuracies:  [0.68359375 0.68359375 0.61328125 0.70703125 0.65882353]


Mean:  0.6692647058823529


Standard Deviation:  0.03187513251380285
```

- **(5)** Use GridSearchCV() to find and print out the best hyperparameter for the number of trees (try the following values: 100, 300, 500, 800, 1000 for 'n_estimator'), ignoring the search for other hyperparameters. Also use 5-fold cross-validation during GridSearchCV. (15 pt)

```
#Grid Parameters
params = {
    'n_estimators': [100, 300, 500, 800, 1000]
}

#Create Instance of GridSearchCV
grid = GridSearchCV(estimator=clf,
                    param_grid=params,
                    cv=5)

#Fit Model
grid.fit(X_train, y_train)
```

```
GridSearchCV(cv=5, error_score=nan,
             estimator=RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
```

```
                                       class_weight=None,
                                       criterion='gini', max_depth=None,
                                       max_features='auto',
                                       max_leaf_nodes=None,
                                       max_samples=None,
                                       min_impurity_decrease=0.0,
                                       min_impurity_split=None,
                                       min_samples_leaf=1,
                                       min_samples_split=2,
                                       min_weight_fraction_leaf=0.0,
                                       n_estimators=300, n_jobs=None,
                                       oob_score=False, random_state=46,
                                       verbose=0, warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'n_estimators': [100, 300, 500, 800, 1000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

```
#Print Best Hyperparameter
best_parameters = grid.best_params_
print("Best Hyperparameter: ", best_parameters)
```

```
Best Hyperparameter:  {'n_estimators': 1000}
```

- **(6)** Find and print out the testing accuracy of the model obtained using the best hyperparameter value on the preprocessed testing dataset (5 pt)

```
best_result = grid.best_score_
print("Best Result: ", best_result)
```

```
Best Result:  0.677092524509804
```

✓ 0s completed at 4:48 PM